UNIVERSITY OF MICHIGAN SCHOOL OF INFORMATION
SI 539: Design of Complex Web Sites

Assignment 6 – A Controller and A View

Due Date: Tuesday February 19, 2008 at 11:55PM

The goal of this assignment is to build your first Rails application and make some changes. You will start with the HTML you produced in Assignment 4. You can keep working on the Rails application you built in Assignment 5 or you can create a new Rails application. The instructions below describe a new Rails application.

Goals

The goal is for you to become familiar with writing view code in a Rails application and to do some very simple controller processing of form parameters. Effectively you will adapt the static HTML from Assignment 4 to become server-produced views.

Steps

The following steps are best done in order with careful testing at the end of each step. Users of Mac OS/X 10.5 can skip steps one and two.

Open a terminal/command window, change into the Rails directory, and run the command to set up your Rails environment:

Windows: **rsu**
Mac OS 10.4: **source rsu.sh**
Mac in the DIAD: **source rsu-diad.sh**

Go into the folder where you develop your Rails applications (typically **rails_apps**)

Make a new rails application:

```
rails assn6 –d sqlite3
```

Go into that application directory and use the Ruby command to generate the code and template files for your controller. The actions (index, members, contact, join, thanks, pictures) are for my htm files - you should make actions that correspond to your htm files. At least one action should be named "index".

```
cd assn6
ruby script/generate controller One index members contact join thanks pictures
```

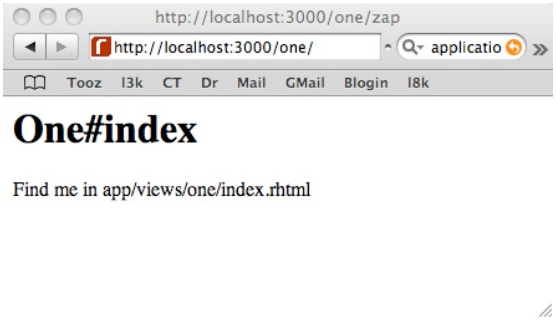
With this command you name your controller “One” and include an action for each of the .htm files in your Assignment 4. In the above example there are six actions (index, members, contact, join, thanks, and pictures. Do not include the .htm suffix in the action names.

Start your application (you should still be in the rails_apps/assn6 folder)

```
ruby script/server
```

Navigate to http://localhost:3000/ -- make sure you get the Welcome to Rails screen.

Navigate to http://localhost:3000/one and you should see a screen that looks as follows:

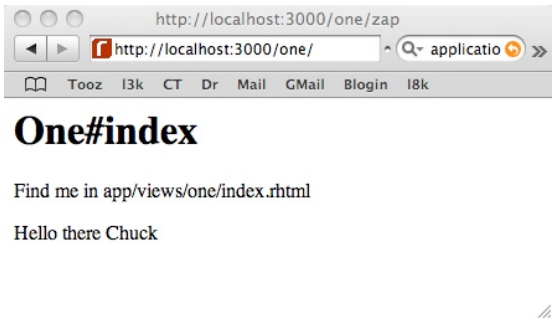


Building your View

In your text editor, open the file index.rhtml in app/views/one – note the "rhtml" file suffix – this means that the file is a Rails HTML template and that you can run "embedded Ruby" in the file. The file should contain two lines with the HTML for the above screen.

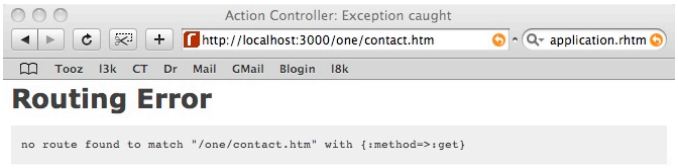
```
<h1>One#index</h1>
<p>Find me in app/views/one/index.rhtml</p>
<p>Hello there Chuck</p>
```

Edit the file and add a paragraph at the end with something silly in it. Save the file and press refresh in your browser. It should look as follows:



Now, copy your index.htm file from Assignment 4, paste it into the index.rhtml file, save the index.rhtml file, and press refresh in your browser. It should look as follows:

You will not see any of your styling – we will fix that later. Press on the Contact link. It should look as follows:



This is OK. When pages come from a Rails template we don't use the same URL conventions to move from one page to another. Open up your index.rhtml file again and change all of the hrefs that navigate between pages as follows:

Old way:
About
Contact

New way:
<a href="**<%= url_for :action => "index" %>**" class="selected">About
<a href="**<%= url_for :action => "contact" %>**">Contact

Note that then only thing we changed (in bold above) was the actual value of the href attribute. We added some "embedded ruby" to the page. Embedded Ruby starts with **<%=** and ends with **%>** -- between these markers you are running code on the server. The **url_for** operation tells Rails to generate a href value to come back to this application with the indicated action. As shown above, do not include the .htm suffix in the action names – actions are not file names. Fix all of the href values in the index.rhtml file, save the file, and go to http://localhost:3000/one/ - you should see the screen with the navigation.

At this point, perhaps you are not seeing the screen with navigation – but instead something like this:

This means that you made an error in your typing in the index.rhtml file. You can look through for hints as to what is wrong – but it is pretty cryptic – at least in the above output it tells you what line the mistake is on.

This would also be a good time to tell you that jEdit can show you line numbers under the **View -> Line Numbers** menu. This will keep you from muttering under your breath as you count lines on the screen.

Error are usually small omissions – like in this case I had typed:

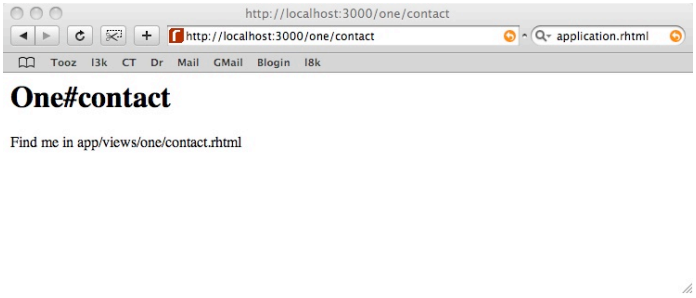
```
<li><a href="<%= url_for :action => "index" >" class="selected">About</a></li>
```

Can you see what is wrong? I ended my embedded Ruby with only a > instead of a %>

```
<li><a href="<%= url_for :action => "index" %>" class="selected">About</a></li>
```

Just keep fixing the file, saving it and pressing refresh until the annoying error finally goes away. Relax – it is usually something really tiny – in time you will be able to find these more quickly – but in the beginning feel free to ask for help with this error.

Assuming you finally got the syntax right, you should see your main screen – now press "Contact" and you should see the following:



This is very good. It means your links are being generated properly.

Now open each of the .rhtml files in your application and paste in the HTML from Assignment 4. Edit the href tags in the <li> elements in all of the files. You should be able to now go to all of your pages and navigate between them.

You will notice that when you press the Submit button you get a message like this:

Routing Error

no route found to match "/one/thanks.htm" with {:method=>:get}

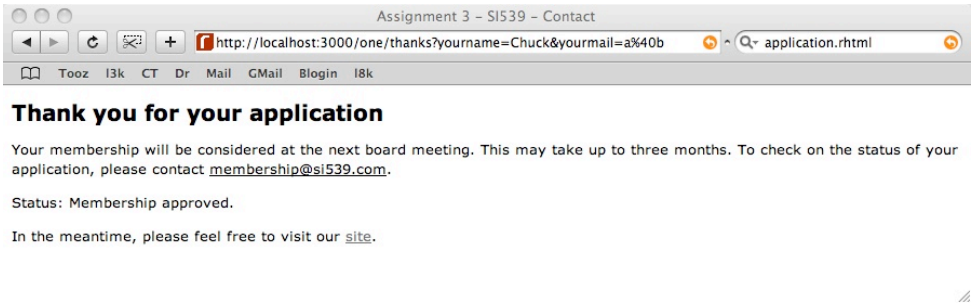
This is because we need to fix the action attribute of the form tag:

Old:
<form method="get" action="**thanks.htm**">

New:

<form method="get" action="**<%= url_for :action => "thanks" %>**">

Once again – we are just replacing the file name with a rails-generated URL that goes to the controller 's thanks action. Once you have made this change and saved it, pressing on the Submit button should look like this:



Notice how the parameters from the form are appended to the URL that the browser gets from the server. We will come back to this later.

Note – do not change the mailto: href – the server is not involved in the mailto: url – this is up to the browser to process.

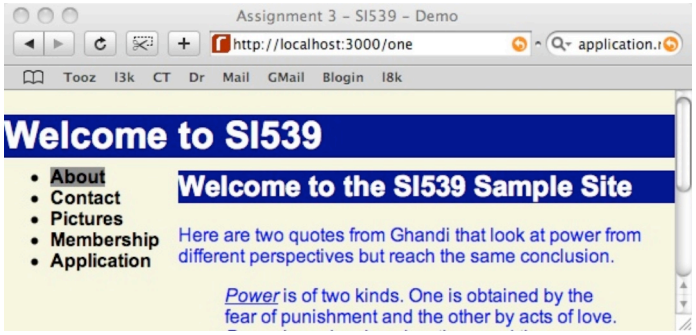
Now we need to get our CSS style sheet working again. Copy your CSS file (in my example I named it style.css) into the folder public/stylesheets – the folder should already exist in your Rails application – if you recall – part of Assignment 5 was to edit index.html in the public folder and make a sub-folder in the public folder. Simply copy your style file into the public/stylesheets folder.

Now go into each of your .rhtml files and change the way that you include the style sheet:

Old:
<link href="style.css" rel="stylesheet" type="text/css" />

New:
<%= stylesheet_link_tag "style.css" %>

Note that we simply replace the entire tag with some Embedded Ruby that will programmatically generate the entire link tag for us. Save these changes and go to http://localhost:3000/one (or press refresh). The output should look as follows:



You should be able to navigate between the screens and they should all be styled nicely.

However your images will still be broken. Copy all of your images to the directory public/images and then change all image tags to the Embedded Ruby as shown below:

Before:

After:
<%= image_tag "ruby-cap-100wide.jpg", :alt => "Ruby cap logo image" %>

Again, like the **stylesheet_link_tag** pattern – the **image_tag** generates the entire img tag.

If your entire site looks right, and you can navigate between all of the pages, and the images all look right, you can move onto the next step of the assignment.

Building Your Controller

In your editor, open the file app/controllers/one_contrtoller.rb – it should look as follows:

```
class OneController < ApplicationController

  def index
  end

  def contact
  end

  [..snip..]
end
```

This is a skeleton controller with one method (entry point) for each of the actions.

So far, we have been using this file all along – when the browser requests a URL like /one/contact – the **contact** action in this file runs (it has no code yet) and then the action looks for the view with the same name as the action with .rhtml appended and shows that view. So the contact action does nothing and then looks for the view called contact.rhtml and sends that back to the browser after processing any Embedded Ruby (stuff between <% and %>) in the view file.

Out next step is to add some code to the thanks action – this is the one that is called when you press the Submit button in the form. We are going to write a simple bit of code to look at one of the parameters and then check the value for one of the parameters and then depending on the value for one of the parameters, change the output on the thanks.rhtml file.

Put the following code into the thanks action:

```
def thanks
  logger.info "Welcome to the thanks action in the controller"
  logger.info params[:yourname]
  if params[:yourname] && params[:yourname] == "Chuck"
     @barcelona = "Membership approved."
  else
     @barcelona = "Membership pending"
  end
end
```

You can pick any secret string for the instant approval – I picked "Chuck". You can also pick any variable name for the data that will be passed to the view from this controller. I picked "barcelona" – you can pick something else. Make sure it has an @ sign as its first character. The @ sign indicates that a variable in Ruby is "special" and is to be passed to the view.

Save this file. Navigate to the form and press Submit. Watch the log file when you do this. In Windows the log is in WinTail and on Macintosh the log is in the terminal window. It should say something like this in the log:

```
Processing OneController#thanks (for 127.0.0.1 at 2008-02-07 18:02:37) [GET]
  Session ID: 878538e199b8de80ee96627a9dc738b2
  Parameters: {"action"=>"thanks", "controller"=>"one", "yourname"=>"Chuck",
"yourmail"=>"a@b"}
Welcome to the thanks action in the controller
Chuck
Rendering one/thanks
Completed in 0.02301 (43 reqs/sec) | Rendering: 0.02081 (90%) | 200 OK
[http://localhost/one/thanks?yourname=Chuck&yourmail=a%40b]
```

This is some very interesting and action-packed stuff – this is why I find it so fascinating to watch the logs while I am building and testing my application. This output shows that we are doing the thanks action in OneController. The Session ID is how Rails keeps each user separate when there are multiple users using the same application at the same time. Then we see the parameters including the parameters from our form (yourname and yourmail). The parameters are what is called a "Hash Map". A Hash Map is like a tiny database – it contains keys to lookup and retrieve information from the hash map. In the above example, "yourname" is a key and "Chuck" is a value. The little "=>" symbol indicates "maps to". So the way to read that in English is as follows:

"Parameters is a hash map where the key yourname maps to Chuck and the key youremail maps to a@b."

The Welcome and Chuck line come from these two lines in your controller:

```
logger.info "Welcome to the thanks action in the controller"
```

```
logger.info params[:yourname]
```

These lines are a way to generate a message in the log when the code in the controller runs – it is a great way to debug your application when things seem not to be going where you thing they are going.

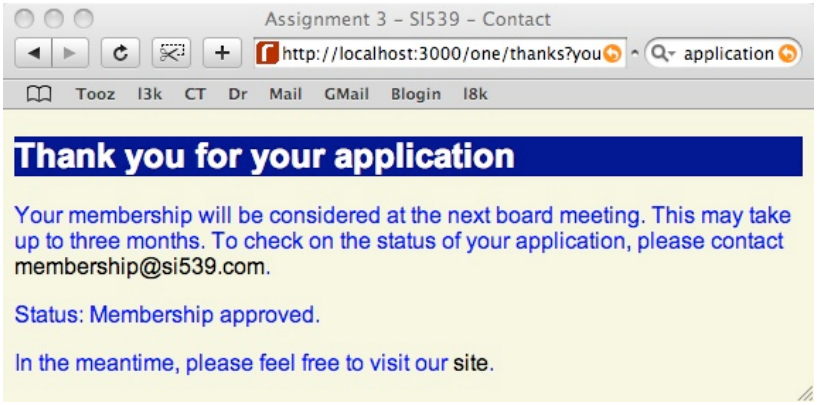
The Rendering line indicates that the action has finished and has passed control to the thanks.rhtml file to generate the output to be sent to the browser.

The next step is to modify your **thanks.rhtml** file and add a line as follows:

```
<p>Your membership will be considered at the next board meeting.
   This may take up to three months.  To check on the status of
   your application, please contact
   <a href="mailto:membership@si539.com">membership@si539.com</a>.</p>
<p>Membership Status: <%= @barcelona %></p>
<p>In the meantime, please feel free to visit our
   <a href="<%= url_for :action => "index" %>">site</a>.</p>
```

The new line contains some text and some Embedded Ruby. The <%= (with an equal sign) says to run the embedded Ruby and print out the results of the Embedded Ruby. It effectively prints out the value in the @barcelona variable. Remember if you chose a variable name to be something other than barcelona, to match the variable names between your controller code and your view template.

Now if you go to your form and press Submit it, it should show you a different member status depending on whether or not the secret string was entered on the form. You should test to make sure that it normally says "Pending" and when you know the secret key – it says "Approved".



Make sure to test with the secret and non-secret values. Make sure to become familiar with the log output so you can see how the processing happens when you go from the form, to the action in the controller to the view.

Hand In

Please hand in the following screenshots:

- A screenshot of http://localhost:3000/one - it should be correctly styled
- A screenshot of your page with the image(s)

- A screenshot of the form page with the values filled in that are not the “secret value”
- A screen shot of the log window after you have pressed submit with the “not secret value”
- A screenshot of the thanks page showing “pending” status
- A screenshot of the thanks page showing the “approved” status